REMIRTA
Ejecución remota de aplicaciones multimedia interactivas de tiempo real

Daniel Espino García

MÁSTER EN INVESTIGACIÓN EN INFORMÁTICA. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTESNE DE MADRID

Trabajo Fin Máster en Ingeniería de Computadores

20 de Junio de 2013

Convocatoria: Julio de 2013

Calificación: Notable

Director:

Eduardo Huedo Cuesta

Autorización de difusión

Daniel Espino García

20 de Junio de 2013

El abajo firmante, matriculado en el Máster en Investigación en Informática de la Facultad de Informática, autoriza a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a su autor el presente Trabajo Fin de Máster: "REMIRTA: Ejecución remota de aplicaciones multimedia interactivas de tiempo real", realizado durante el curso académico 2012-2013 bajo la dirección de Eduardo Huedo Cuesta en el Departamento de Arquitectura de Computadores y Automática, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Resumen en castellano

REMIRTA es una herramienta para la ejecución remota de aplicaciones multimedia interactivas de tiempo real. Sigue un esquema de cliente/servidor en el cual el servidor ejecuta toda la carga de trabajo de la aplicación, y el cliente recibe a través de streaming las imágenes que genera. El proyecto se ha desarrollado con el lenguaje de programación Python, y busca la optimización de la latencia entre el servidor y el cliente. La principal aplicación de este proyecto es en la industria de los videojuegos, pero también se puede extender a otras aplicaciones como los escritorios remotos.

Palabras clave

Cloud, Multimedia, tiempo real, videojuego, remoto, interactivo, streaming, Cloud Gaming

Abstract

REMIRTA is a tool developed for the remote execution of multimedia interactive real-time applications. It follows a client/server model, where the server executes all the workload of the application, while the client receives by streaming the generated images. The project has been developed with the Python programming language, and focus on the optimization of the latency between the client and the server. The main application of this project is on the videogames industry, but it can be used for other applications, like remote desktops.

Keywords

Cloud, Multimedia, real-time, videogames, remote, interactive, streaming, Cloud Gaming

Índice General

Autorizacion i

Resumen ii

Abstract iii

Índice iv

Índice Figuras vii

Índice de Tablas ix

Agradecimientos xi

1. Introducción 1

2. Estado del arte 3
2.1. Plataformas 4
2.1.1. OnLive 4
2.1.2. Gaikai 4
2.1.3. Ubitus 5
2.1.4. Gaming Anywhere 5
2.2. Protocolos 6
2.2.1. MPEG4 6
2.2.2. RFB 7
2.2.3. REPRO 7

3. Herramientas 9

4. Arquitectura 11
4.1. Cliente 11
4.2. Servidor 13
4.3. Protocolo 15

5. Evaluación 18
5.1. Entorno de prueba 18
5.2. Resultados 19

6. Conclusión **26**

7. Trabajo futuro **27**

Bibliografía **28**

A. Métodos de compresión de flujo **30**
A.1. Sin compresión . . . 30
A.2. Detección de cambios a nivel de píxel . . . 30
A.3. Detección de cambios en línea . . . 30
A.4. Detección de cambios en línea discontinua . . . 31
A.5. Detección de movimientos de tipo desplazamiento . . . 31
A.6. Cambio significativo . . . 31
A.7. Cambio significativo robusto . . . 31

B. Mensajes en REPRO **33**
B.1. Mensaje de envío de eventos . . . 33
B.1.1. QUIT . . . 34
B.1.2. VIDEOEXPOSE . . . 34
B.1.3. ACTIVEEVENT . . . 34
B.1.4. KEYDOWN . . . 34
B.1.5. KEYUP . . . 35
B.1.6. MOUSEBUTTONDOWN . . . 35
B.1.7. MOUSEBUTTONUP . . . 35
B.1.8. MOUSEMOTION . . . 35
B.1.9. VIDEORESIZE . . . 36
B.2. Mensaje de envío de imágenes . . . 36
B.2.1. Sin compresión . . . 36
B.2.2. Sólo cambios . . . 36
B.2.3. Cambios en línea . . . 37
B.2.4. Cambio por desplazamiento . . . 37

Índice de Figuras

2.1. Logo de OnLive 4
2.2. Logo de Gaikai 4
2.3. Logo de Ubitus 5
2.4. Logo de Gaming Anywhere 5
2.5. Logo de H.264 6
2.6. Logo de Real VNC, creadores de RFB 7
2.7. Logo provisional de REPRO 7

3.1. Esquema de funcionamiento de un programa de PyGame 10

4.1. Esquema de la arquitectura utilizada 12
4.2. Esquema de comportamiento del cliente 13
4.3. Esquema de comportamiento del servidor 16

5.1. Evolución del tamaño de los mensajes de imágenes a través del tiempo. 20
5.2. Evolución del tamaño de los mensajes de eventos a través del tiempo. 20
5.3. Tiempo de proceso de cada mensaje de imágenes para MaTris master. 21
5.4. Tiempo de proceso de cada mensaje de eventos para Matris master. . 22
5.5. Tiempo de proceso de cada mensaje de imágenes para Lunar Panda. . 22
5.6. Tiempo de proceso de cada mensaje de eventos para Lunar Panda. . . 23
5.7. Tamaño de las imágenes para los tres algoritmos. 24
5.8. Tiempo de proceso de las imágenes para los tres algoritmos. 25

Índice de Tablas

5.1. Especificación del equipo Acer Aspire 5610 18
5.2. Especificación del equipo Toshiba Satellite L750 19
5.3. Medidas sobre los tamaños de los datos 21
5.4. Medidas sobre los tiempos de codificación y decodificación de los datos 23

Agradecimientos

Agradezco a los profesores de la asignatura de Computación en Red y Tecnologías Grid por inspirarme para este proyecto, a mi hermano por su incansable asesoramiento, y a mis antiguos compañeros por su apoyo incondicional.

Capítulo 1

Introducción

En el mundo actual tenemos gran cantidad de dispositivos con prestaciones limitadas: teléfonos inteligentes, netbooks, tablets, portátiles... Estos dispositivos nos proveen la movilidad que otros dispositivos más potentes, como pueden ser los ordenadores de sobremesa, no nos permiten. Esto llega hasta el punto en que muchas veces, incluso en los hogares, se prefiere usar alguno de estos dispositivos desde el sofá del salón, antes que usar el ordenador de sobremesa.

Por otro lado, la industria del entretenimiento interactivo (entiéndase, videojuegos) crea cada vez nuevos programas más exigentes sobre las capacidades de los dispositivos. Incluso las aplicaciones diseñadas para plataformas móviles, en su evolución, dejan obsoletos terminales antiguos. De esta manera, para disfrutar de estas experiencias, es necesario actualizar el dispositivo por uno de mejores características cada poco tiempo.

Además, hay una serie de procesos bastante incómodos a la hora de adquirir una aplicación de este tipo. Por ejemplo:

- Encontrar la aplicación (en tienda física o en tienda de descargas)
- Encontrar una versión de prueba para comprobar si la aplicación cumple las expectativas
- Descargarla en caso de tienda online
- Instalarla

- El coste en espacio en el ordenador
- Comprobar si tu dispositivo realmente cumple con las características necesarias para ejecutar la aplicación

Si tenemos en cuenta el coste de estos procesos, el uso de estas aplicaciones puede convertirse en una odisea.

Y no sólo eso. Una vez instalado en nuestro dispositivo de sobremesa, siempre tenemos que usar ese dispositivo para usar nuestra aplicación. ¿Por qué no poder usarla desde nuestra tablet, o desde nuestra SmartTV?

Nuestra aplicación, REMIRTA (de las siglas en inglés "Remote Execution of Multimedia Interactive Real-Time Applications"), es una solución a todos los puntos previamente mencionados. Un sistema que ejecute la aplicación en un dispositivo remoto (podría ser nuestro dispositivo de sobremesa en nuestras casas, o incluso un dispositivo en la nube), y que comunique toda la información necesaria con otro dispositivo.

De esta manera, una vez instalado (si se encontrara en la nube, el mismo proveedor se encargaría de tener los programas previamente instalados y actualizados), podemos acceder al juego con cualquier dispositivo. Tan sólo necesitaríamos una conexión (ya sea inalámbrica, cableada, o por internet). Además, siendo la ejecución en el equipo remoto, el equipo final no necesita grandes prestaciones, ya que tan sólo se dedicará a reproducir el flujo de imágenes.

Además, centrándonos en el sistema usando la nube, permite varias mejoras para la industria. Al no llegar en ningún momento el programa al usuario final, se reduce la posibilidad de piratería. Por otro lado, en caso de pagar por uso, se propicia el desarrollo de aplicaciones de larga duración, o con grandes posibilidades de reutilización. Por último, a nivel de control parental, es más fácil controlar cuánto tiempo pasan los hijos jugando y a qué, debido al pago por uso y al historial de juego.

Finalmente, este proyecto se ha desarrollado como software libre. El objetivo es que tanto el colectivo social como académico puedan estudiar y contribuir a esta plataforma para un desarrollo futuro.

Capítulo 2

Estado del arte

La idea que proponemos tiene actualmente gran impacto en la industria. Tras un estudio previo, se ha comprobado que varias empresas ya están haciendo aplicaciones similares. Además, toda la comunicación y la compresión de las imágenes tienen que estar sujeta a un protocolo. Posteriormente, en este capítulo, nos dispondremos a presentar algunas de estos programas y protocolos.

En la industria de los videojuegos, se tiende al uso de Cloud Gaming. Se basa en la ejecución del juego en la nube. Aun así, no hemos encontrado publicaciones a nivel académico, o público que profundicen en el tema. Desde la Academia Sinica, en Taiwán, investigadores han realizado estudios sobre cómo medir la latencia en Cloud Gamign[2], o las diferencias entre distintos juegos, y su capacidad para funcionar en Cloud Gaming[1]. Estos mismos autores son los creadores de GamingAnywhere, una plataforma libre para Cloud Gaming. Cabe destacar que GamingAnywhere fue lanzada a principios del 2013, posteriormente al inicio de este proyecto.

2.1. Plataformas

2.1.1. OnLive

Figura 2.1: Logo de OnLive

OnLive[6] es una compañía que lleva desde 2010 sirviendo juegos en la nube. Desde su plataforma son capaces de servir actualmente a una gran variedad de dispositivos, incluidos PCs, MACs, televisores, o teléfonos inteligentes. Aun así no ofrecen ninguna información sobre la tecnología que usan. El modo de uso es la descarga de una aplicación cliente, que se conecta con el servidor, y ofrece los juegos a través de streaming.

2.1.2. Gaikai

Figura 2.2: Logo de Gaikai

Gaikai[7] permitía usar a través de navegador juegos en formato de streaming. Para esto usaba herramientas como Java, o Adobe Flash. Así se podía incrustar los juegos en diferentes portales web, de manera que no se necesitaba utilizar ningún programa aparte, ni registrarse, ni ir a otra página web para jugarlo. De este modo se podía probar una demonstración, y una vez acabara el tiempo de prueba, te

ofrecía que compraras el juego, o siguieras utilizándolo en la nube, pagando por uso. Recientemente la plataforma ha sido comprada por Sony Computer Entertainment. En la edición de Junio de 2013 de la E3 (Electronic Entertainment Expo) se desveló que la siguiente generación de consolas de Sony (Play Station 4) contaría con la participación del equipo de Gaikai para ofrecer los juegos de esta consola, y otras consolas previas a través de la tecnología de Cloud Gaming[10].

2.1.3. Ubitus

Figura 2.3: Logo de Ubitus

Ubitus[8], al igual que Gaikai, permite la ejecución de juegos a través de navegador. Respecto a su tecnología, en su página web se puede encontrar que tienen soporte para protocolos de streaming como MPEG4 y H.264, varios servidores iRTSP (internet Real Time Streaming Protocol) y tecnología de streaming pendiente de patente, entre otras características.

2.1.4. Gaming Anywhere

Figura 2.4: Logo de Gaming Anywhere

Gaming Anywhere[9] es una aplicación libre para soportar Cloud Gaming. Es un sistema cliente servidor, en el que el cliente se conecta a un portal, en el que elige el juego. El portal se encarga entonces de configurar el juego, y establecer la conexión

entre el usuario y el juego. Todo su código es abierto y está disponible para varias plataformas.

2.2. Protocolos

Todas estas plataformas se plantean sobre distintos protocolos de comunicación entre cliente y servidor. A continuación se detallan distintos protocolos que pueden usarse para estas herramientas.

2.2.1. MPEG4

Figura 2.5: Logo de H.264

MPEG4[5] es un método de compresión digital de audio y vídeo. Este sistema es muy amplio, e incluye varios estándares que son denominados "partes". Cada parte se centra en un aspecto (audio, video, software de referencia,...), siendo la parte 10 la que está más relacionada con este proyecto.

La parte 10 también es conocida como H.264. Es uno de los estándares de compresión de vídeo más utilizados actualmente para la distribución de vídeos en alta definición. Entre otras cosas es el estándar utilizado en los discos Blu-Ray.

Este protocolo, al igual que otros protocolos dentro de MPEG4, es muy potente, pero su carácter cerrado nos desanima a su uso en este proyecto, que busca crear una aplicación libre.

2.2.2. RFB

Figura 2.6: Logo de Real VNC, creadores de RFB

RFB[4] (Remote Frame Buffer) es un protocolo simple para acceso remoto a interfaces gráficas. Fue desarrollado por el laboratorio de investigación Olivetti, y más tarde ganó más fuerza con el programa VNC (Virtual Network Computing). Al cerrarse el laboratorio Olivetti, los creadores de VNC crearon la RealVNC. Dese esta empresa, se encargan de continuar el desarrollo de VNC y mantener el estándar RFB. En su página web se puede encontrar la especificación del protocolo.

Este protocolo trabaja a nivel de "frame buffer", lo que le permite ser utilizado en cualquier sistema de ventanas.

Este sistema se acerca mucho al sistema que queremos crear, pero ha sido desestimado para dar un protocolo más especializado en el problema en el que nos encontramos.

2.2.3. REPRO

Figura 2.7: Logo provisional de REPRO

Nosotros proponemos un protocolo propio para REMIRTA: REPRO (REMIRTA Protocol). Los aspectos más básicos de este protocolo se pueden encontrar en la

sección 4.3. Para una visión más detallada de los algoritmos de compresión de flujo remítase al apéndice A. Para una visión más profunda sobre la estructura de los mensajes acuda al apéndice B.

Capítulo 3

Herramientas

El lenguaje de programación utilizado para este proyecto ha sido Python. Éste es un lenguaje de alto nivel, interpretado, y de gran extensión. Se ha decidido utilizar Python por la rapidez y sencillez que permite en el desarrollo.

Junto con este lenguaje se ha utilizado PyGame[3], un conjunto de módulos para Python diseñados para escribir juegos. PyGame otorga una interfaz sobre las librerías de SDL (Simple DirectMedia Layer) junto con otras funcionalidades. Se ha decidido utilizar este módulo por estar escrito en Python, y debido a que la cercanía con SDL podrá permitir, en un futuro, una reescritura del código más cómoda.

La estructura de un programa en PyGame tiene unos aspectos clave. Al principio del programa se inicializan las distintas variables, y se inicializa el "display". El "display" funcionará como superficie donde dibujaremos el juego. Una vez terminadas todas las inicializaciones, se comienza con el bucle de juego. Según la tasa de refresco a la que se quiera que corra el juego, se le debería asignar a cada iteración del bucle una fracción de tiempo. Dentro del bucle de juego, se suelen incluir cuatro secciones:

- Control de la entrada
- Actualización de estado
- Redibujado del "display"
- Refrescado del "display"

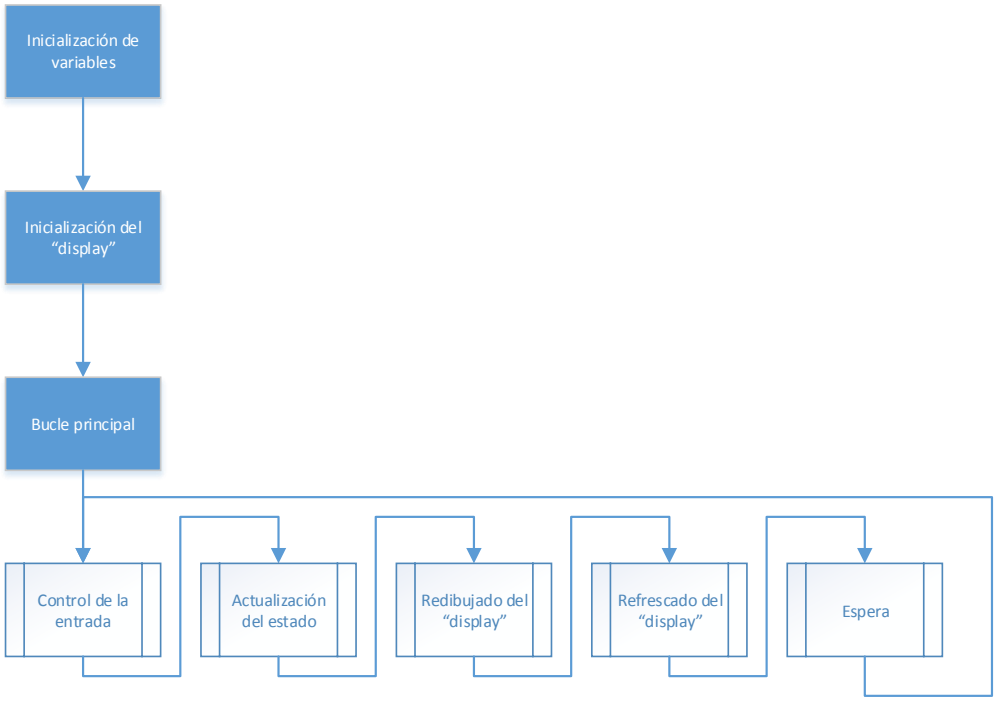

Figura 3.1: Esquema de funcionamiento de un programa de PyGame

Tras redibujar el display, una práctica común es esperar hasta agotar la fracción de tiempo, y así no superar la tasa de refresco deseada.

En la figura 3.1 se puede ver un esquema de un programa típico de PyGame.

Además de estas dos herramientas se han implementado algunas funciones en C. Esto ha sido necesario porque el lenguaje Python tiene limitaciones de rendimiento ante bucles grandes. También se han implementado las interfaces correspondientes entre C y Python.

Por último, para todos los modelos matemáticos y trabajo con los resultados se ha usado la herramienta Matlab.

Capítulo 4

Arquitectura

La arquitectura usada en nuestra solución es un modelo cliente-servidor. Además, proponemos un protocolo de mensajes entre éstos. En la figura 4.1 se puede encontrar un esquema general de la arquitectura.

4.1. Cliente

El cliente consta de:

- El programa principal
- El hilo de envío
- El hilo de recepción

El programa principal sigue el siguiente proceso:

1. Inicia los hilos
2. Inicia el "display"
3. Ejecuta el bucle principal
    a) Refresca el "display"

Tanto el hilo de recepción como el de envío están conectados al servidor, y se encargan de la comunicación entre el cliente y el servidor. El hilo de recepción se encarga de redibujar el "display" siguiendo el siguiente proceso:

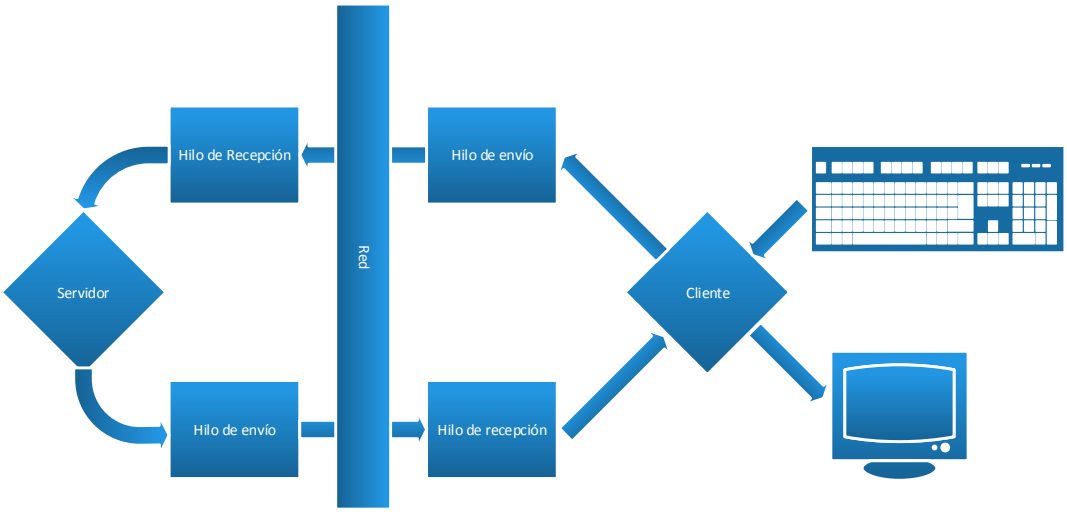

Figura 4.1: Esquema de la arquitectura utilizada

1. Carga los primeros bytes del mensaje de entrada
2. Decide qué tipo de mensaje está recibiendo
3. Lo decodifica
4. Termina de cargar el resto del mensaje
5. Aplica los cambios sobre el "display"

Por otro lado, el hilo de envío, se encarga de captar las pulsaciones del teclado del usuario, y enviarlos al servidor. El proceso que sigue es el siguiente:

1. Toma los últimos eventos producidos por el usuario
2. Codifica los eventos en un mensaje
3. Envía el mensaje
4. Espera un tiempo prudencial

El tiempo prudencial corresponde a consumir una sección de tiempo. La sección que se ha usado en las pruebas es de 17ms (60Hz). Si este tiempo fuera mucho mayor,

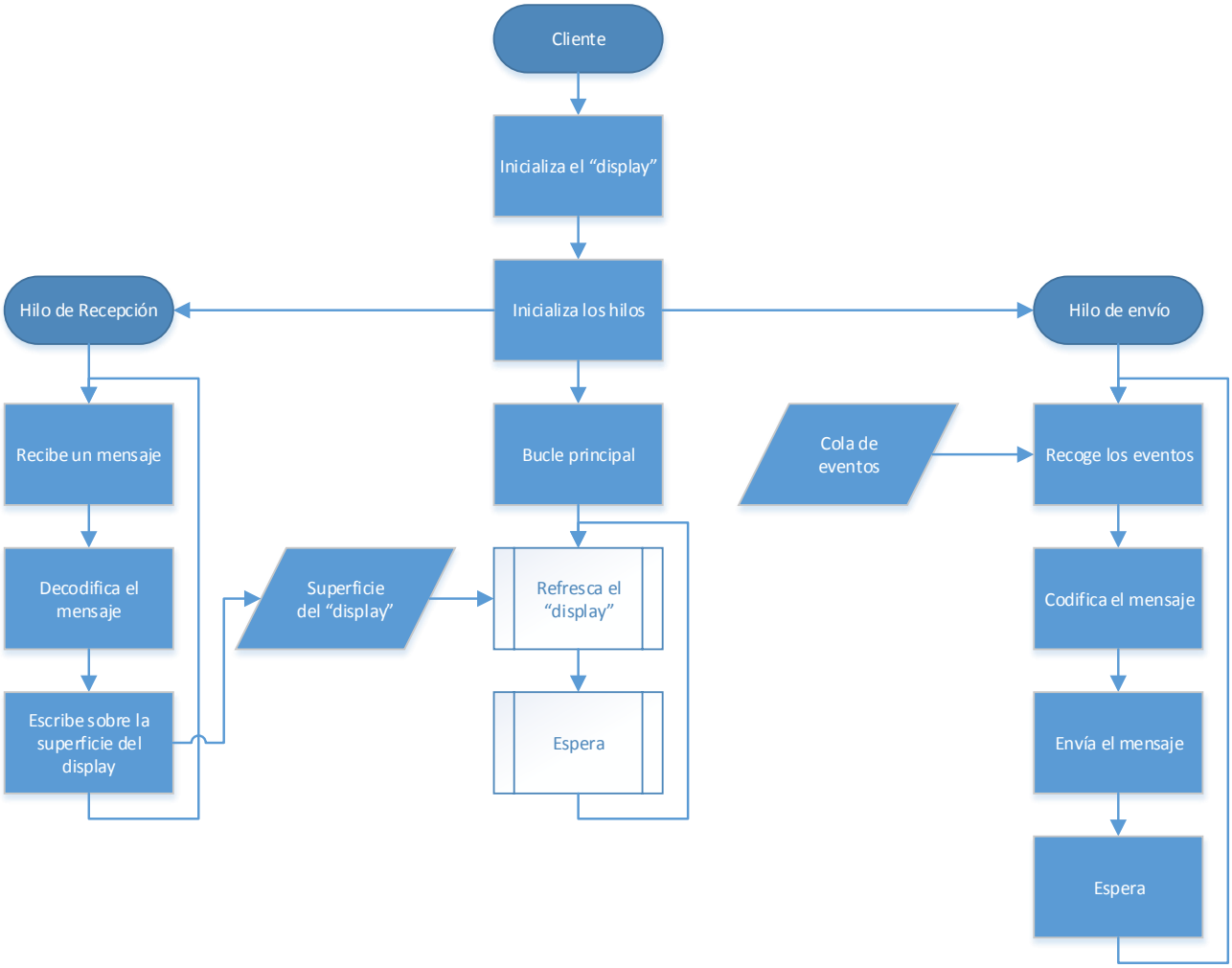

Figura 4.2: Esquema de comportamiento del cliente

la velocidad de respuesta del juego sería muy baja. Sin embargo, si el tiempo fuera muy corto, saturaríamos la red de mensajes muy pequeños.

En la figura 4.2 podemos encontrar un esquema del funcionamiento.

4.2. Servidor

En esta primera versión del sistema, la aplicación estará integrada en el juego. Resulta necesario hacer unos cambios bastante pequeños en el código del juego original para que toda la información sea enviada a través de la red, en vez de ser

enviada a los dispositivos habituales de entrada y salida. Los cambios a hacer son los siguientes:

- Inicializar los hilos y demás variables relativas a REMIRTA.
- Cambiar la inicialización del "display" por la inicialización de una superficie sobre la que escribirá el juego.
- Cambiar cualquier acción relacionada con el "display" por su correspondiente función de REMIRTA.

Una vez realizados los cambios, tendremos tres procesos distinguidos:

- El programa principal
- El hilo de envío
- El hilo de recepción

El programa principal será el que se encargue de ejecutar el juego. Además, para mantener su velocidad sin detrimento por la velocidad de transmisión, se utilizan dos superficies distintas: una para el uso del juego, y otra para el uso del hilo de envío. Cada vez que se reescribe la superficie de juego, se sigue el siguiente proceso:

1. Se comprueba si está tomado el bloqueo sobre la superficie de envío
    a) Si está tomado, se continúa la ejecución del juego
    b) Si no está tomado, se toma
        1) Se copia la superficie de juego sobre la superficie de envío
        2) Se marca esta superficie como lista para enviar
        3) Se libera el bloqueo

El hilo de envío es el encargado de enviar los diferentes mensajes relacionados con la imagen al cliente. El proceso que ejecuta este hilo es el siguiente:

1. Comprueba si la superficie de envío está lista para enviar

   a) Si está lista, toma el bloqueo sobre la superficie de envío

   b) Sigue el protocolo, y crea el mensaje

   c) Envía el mensaje al cliente

2. Espera un tiempo prudencial

Este tiempo prudencial del que se habla, tiene la misma función que el tiempo prudencial usado en el envío de eventos en el cliente. En este caso se ha usado 40 ms (25Hz). En este caso, al poner el número más bajo, por el mayor tamaño de los mensajes, generaría un tráfico inadmisible. Sin embargo, usando un tiempo mayor, podría llegar a una tasa de refresco tan baja que no llegara a generar la ilusión de movimiento.

Por último, el hilo de recepción se encarga de la recepción de los eventos generados por el usuario. Su proceso es el siguiente:

1. Recibe el mensaje
2. Decodifica la longitud del mensaje
3. Termina de cargar el mensaje
4. Decodifica el mensaje
5. Añade todos los eventos a la lista de eventos del juego

En la figura 4.3 se puede encontrar un esquema del funcionamiento del servidor.

4.3. Protocolo

Como dijimos anteriormente, hemos decidido denominar al protocolo REPRO (REMIRTA Protocol). Este protocolo intenta reducir al mínimo el tamaño de los mensajes. Para conseguir esto, trabajamos a nivel de bit. De esta manera, reducimos aún más el tamaño del mensaje.

Junto con el protocolo se usa una serie de técnicas para reducir el flujo de mensajes. En una visión muy ingenua, podemos resolver el problema enviando cada imagen

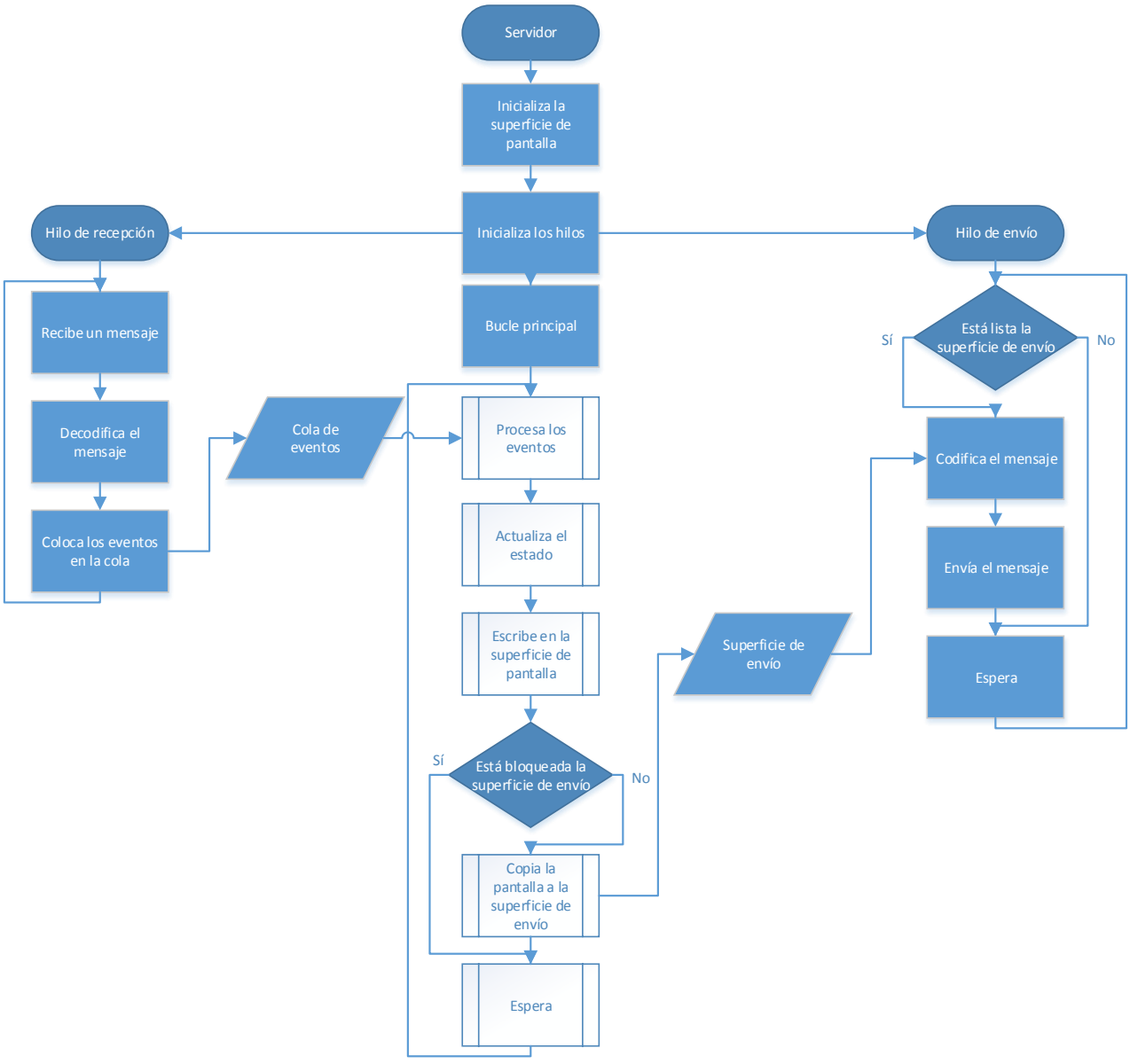

Figura 4.3: Esquema de comportamiento del servidor

completa a través de la red. Para una pantalla de resolución de 800x600, con 16 millones de colores, y una tasa de refresco de 25Hz, tendríamos que enviar 36MB por segundo. Esto es una cantidad muy elevada teniendo en cuenta las redes actuales. Además, las aplicaciones suelen funcionar a resoluciones mayores y con mayor tasa de refresco.

Además, al ser una aplicación interactiva, las imágenes se van generando según se van sirviendo. Por esto no podemos hacer algoritmos basados en futuros cambios, como algunos sistemas de compresión.

Los algoritmos probados han sido el algoritmo sin compresión, el de cambios píxel a píxel, y el de cambios por líneas. Otros algoritmos implementados y propuestos, y la especificación de estos se pueden encontrar en el apéndice A.

Una visión más profunda de los mensajes utilizados en el protocolo se puede encontrar en el apéndice B.

Capítulo 5

Evaluación

5.1. Entorno de prueba

Para la realización de este trabajo se ha usado un ordenador portátil Acer Aspire 5610 como servidor. El sistema operativo usado ha sido una distribución de Linux Lubuntu 12.04. Las especificaciones de este equipo se encuentran en la tabla 5.1.

Para las pruebas con ordenador remoto, se ha utilizado un portátil Toshiba Satellite L750 como cliente. El sistema operativo usado ha sido una distribución de Linux Lubuntu 12.10 sobre una máquina virtual de Oracle VM VirtualBox, en un Windows 7 Home Premium de 64bits. Las especificaciones de este equipo se encuentran en la tabla 5.2.

La comunicación se ha establecido a través de una red inalámbrica usando un router VR-3035u.

Las aplicaciones utilizadas para las pruebas han sido recogidas de la página web

Categoría	Especificación
Procesador	Intel Core Duo processor T2300 2MB L2 cache
Núcleo	Intel 945GM Express
RAM	1GB (2*512) DDR2 533 MHz soDIMMs
Sistema de video	Intel 945GM gráficos 3D integrados, con Intel Graphics Media Accelerator (GMA) 950 y hasta 224 MB de memoria compartida
Ethernet	WLAN: Intel PRO/Wireless 3945ABG dual-band tri-mode 802.11a/b/g

Tabla 5.1: Especificación del equipo Acer Aspire 5610

Categoría	Especificación
Procesador	Intel Pentium processor B960 2.20 GHz L3 Cache : 2 MB
RAM	4,096 MB DDR3 RAM (1,333 MHz)
Adaptador gráfico	NVIDIA GeForce GT 520M con Tecnología CUDA Memoria: 1024 MB de VRAM dedicada Tipo de memoria: DDR3 Video RAM Conexión del bus: PCI Express
Comunicación sin cables	Compatibilidad: Wi-Fi Soporte de red: 802.11b/g/n Fabricante: Atheros Tecnología inalámbrica: Wireless LAN

Tabla 5.2: Especificación del equipo Toshiba Satellite L750

de PyGame, aportadas por la comunidad. El primero de los juegos utilizados es "MaTris master" (Smart Viking), un clon de Tetris usando PyGame. También se ha usado "Lunar Panda" (Gimpy Software), un juego en el que tenemos que controlar a un panda, con un cohete, que desea aterrizar en la luna.

5.2. Resultados

En las primeras pruebas hemos utilizado el equipo Acer para que funcione de servidor y de cliente a la vez. En las figuras 5.1 y 5.2 se pueden ver la evolución respecto al tiempo del tamaño de los mensajes. Como método de compresión se ha usado la una solución mixta entre imágenes sin comprimir y compresión basada en cambios en línea.

En la primera gráfica podemos observar altos picos de envío en ciertos momentos debido a los cambios que se efectúan cuando hay un cambio por completo de pantalla. Por otro lado los mensajes relativos a los cambios son mucho más pequeños en relación. También se puede observar que en Lunar Panda, al haber menor cantidad de movimientos, se tiene unas tasas muy buenas. En la tabla 5.3 se pueden observar una serie de medidas respecto a estos valores.

Teniendo en cuenta estos valores, si tomamos la frecuencia de envío de imágenes que teníamos en el capítulo 4.2 de 25Hz, vemos que necesitaríamos para reproducir

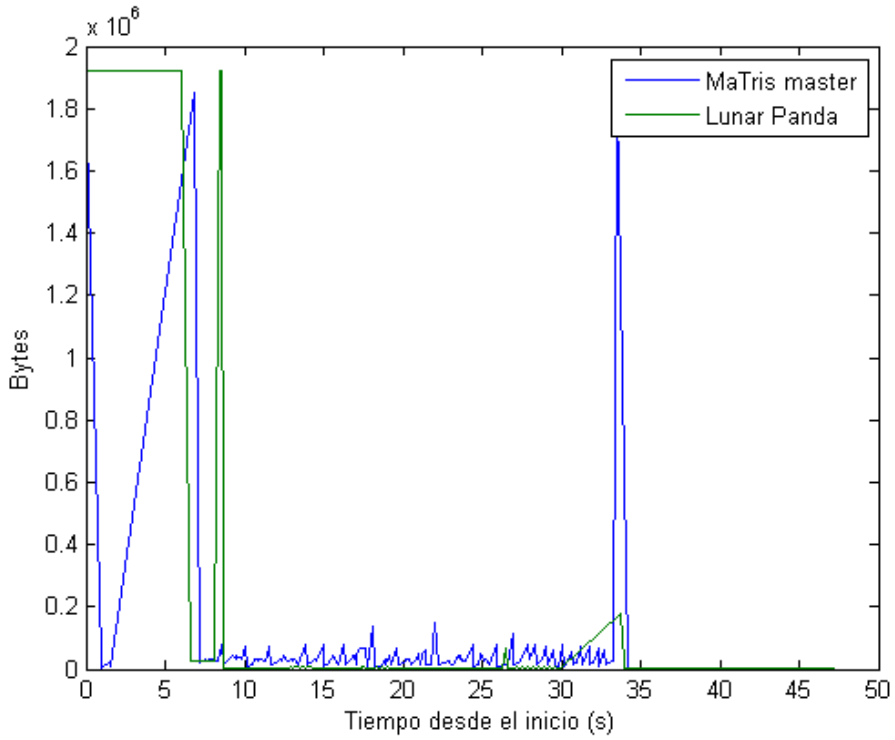

Figura 5.1: Evolución del tamaño de los mensajes de imágenes a través del tiempo.

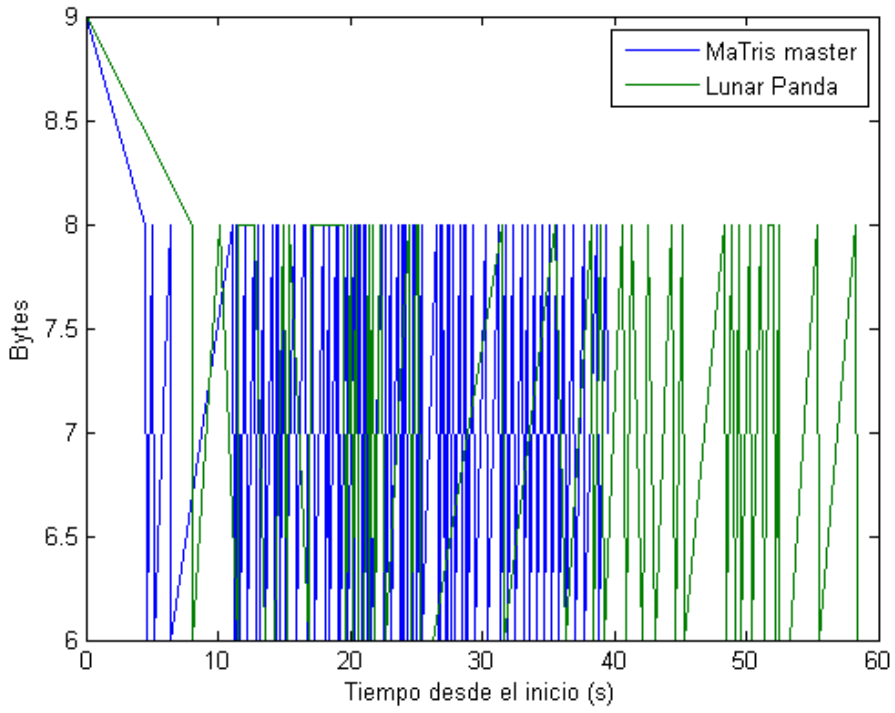

Figura 5.2: Evolución del tamaño de los mensajes de eventos a través del tiempo.

	MaTris master		LunarPanda	
	Imagen	Eventos	Imagen	Eventos
Mínimo	4.778B	6B	40B	6B
Media	88.086B	7B	36.708B	7B
Máximo	1.848.005B	9B	1.920.005B	9B

Tabla 5.3: Medidas sobre los tamaños de los datos

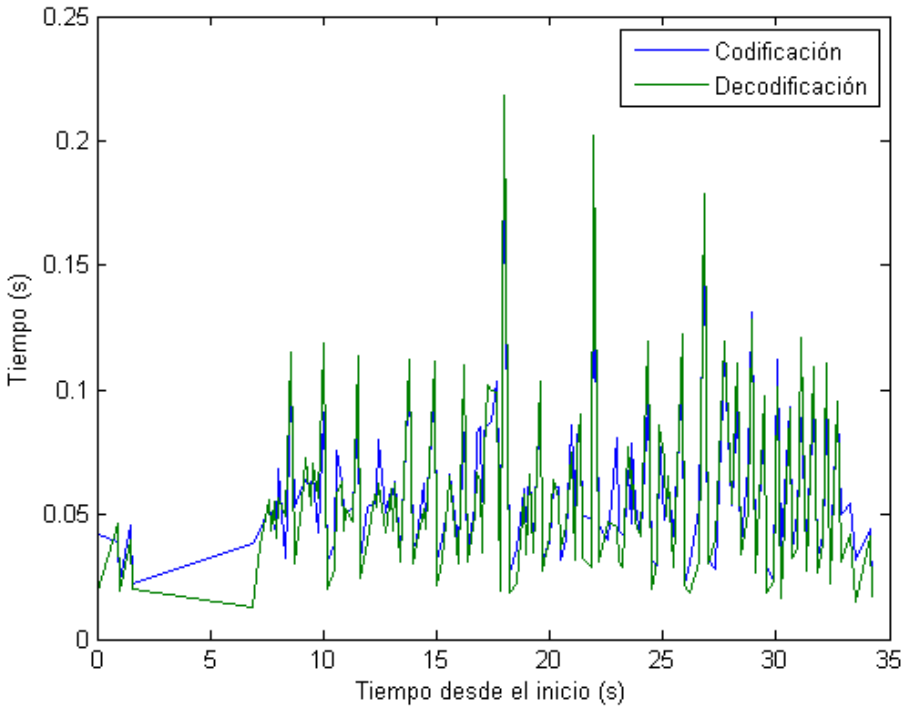

Figura 5.3: Tiempo de proceso de cada mensaje de imágenes para MaTris master.

este sistema una conexión de 2.2MB/s en el caso de MaTris master, y 0.9MB/s en el caso de Lunar Panda. Estos datos son muy prometedores, porque son accesibles por la tecnología de la red de datos actual. Además, con futuras optimizaciones se espera poder reducir más estos valores.

En las figuras 5.3, 5.4, 5.5 y 5.6 se puede observar el tiempo que ha tomado para cada juego tanto codificar como decodificar el mensaje.

Aunque en la gráfica no se pueda apreciar claramente, el tiempo de decodificación es por norma general menor, debido a su menor carga de trabajo. Además vemos que por norma general la carga de trabajo no es excesivamente alta en las imágenes, y despreciable en la mayoría de los casos en los eventos. Vemos que para Lunar Panda

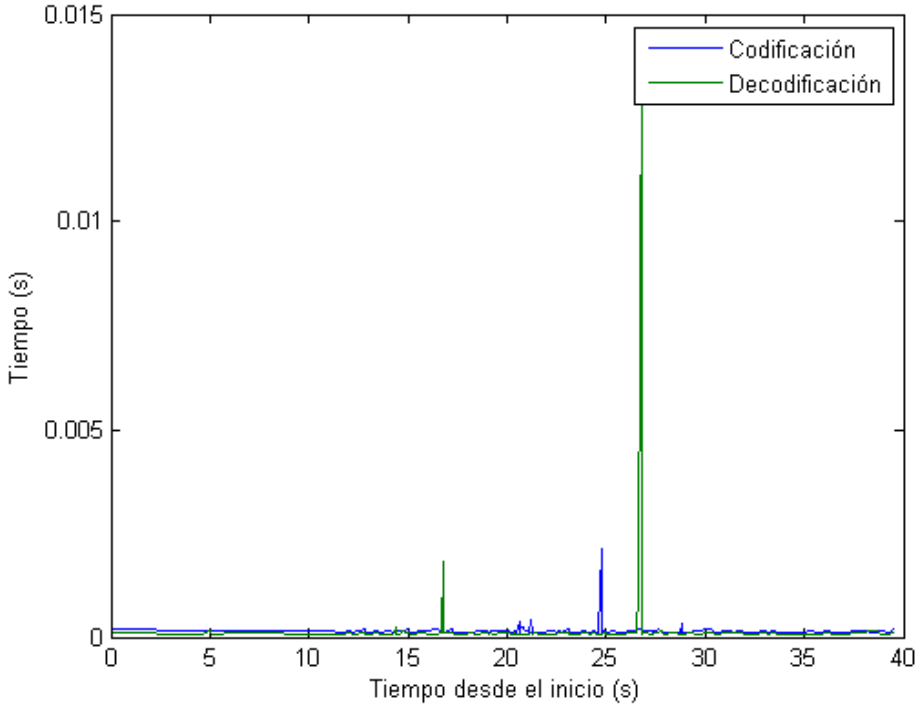

Figura 5.4: Tiempo de proceso de cada mensaje de eventos para Matris master.

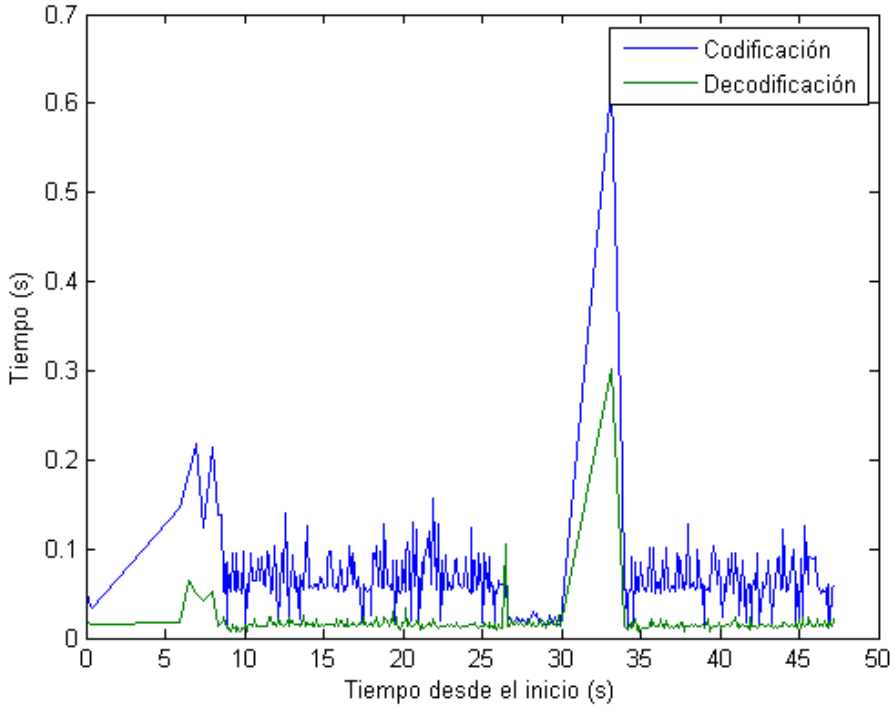

Figura 5.5: Tiempo de proceso de cada mensaje de imágenes para Lunar Panda.

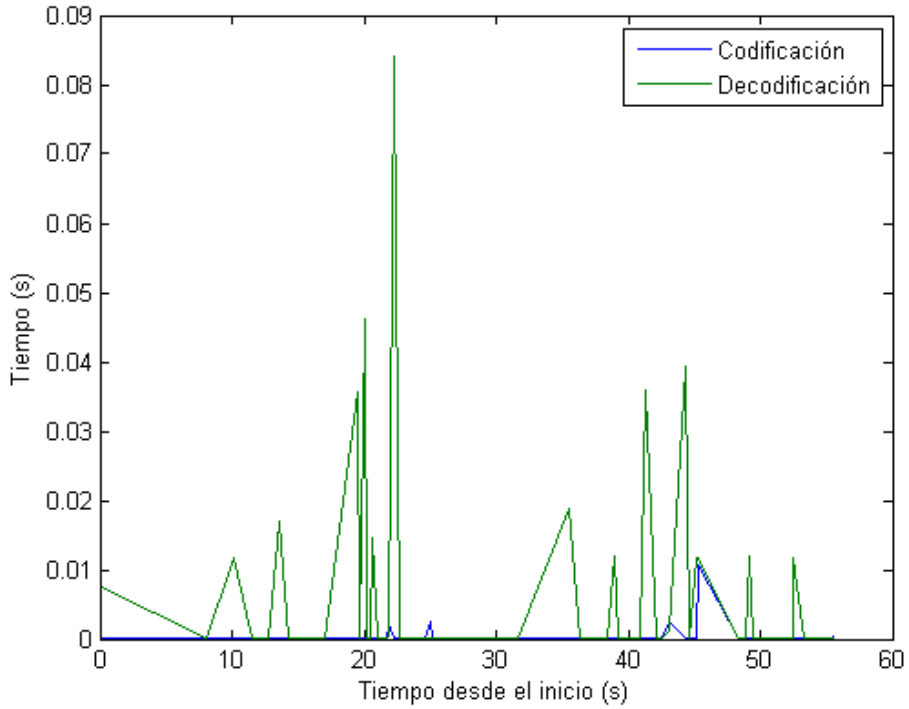

Figura 5.6: Tiempo de proceso de cada mensaje de eventos para Lunar Panda.

	MaTris master				Lunar Panda			
	Imagen		Eventos		Imagen		Eventos	
	Cod.	Dec.	Cod.	Dec.	Cod.	Dec.	Cod.	Dec.
Mínimo	19ms	12ms	0.09ms	0.06ms	13ms	8ms	0.07ms	0.04ms
Media	59ms	56ms	0.2ms	0.2ms	63ms	17ms	0.4ms	5.5ms
Máximo	190ms	218ms	2.1ms	14.5ms	621ms	301ms	10ms	84ms

Tabla 5.4: Medidas sobre los tiempos de codificación y decodificación de los datos

se disparan algunas métricas. Esto se debe a que la complejidad de los algoritmos físicos que usa requiere mayor cómputo, retrasando la ejecución de otras tareas. En la tabla 5.4 se pueden apreciar unas medidas de estos valores.

Si comparamos con los datos que aportan en el artículo [2], también encontramos resultados muy esperanzadores. Lo que ellos denominan Processing Delay, se correspondería con el tiempo de decodificación de un evento más la codificación de una imagen. Por otro lado, el Playout delay correspondería a la decodificación de la imagen. En comparación, frente a los 200ms de Processing Delay que se consiguen en OnLive, nosotros conseguimos tiempos de, aproximadamente, 70ms. Por otro lado,

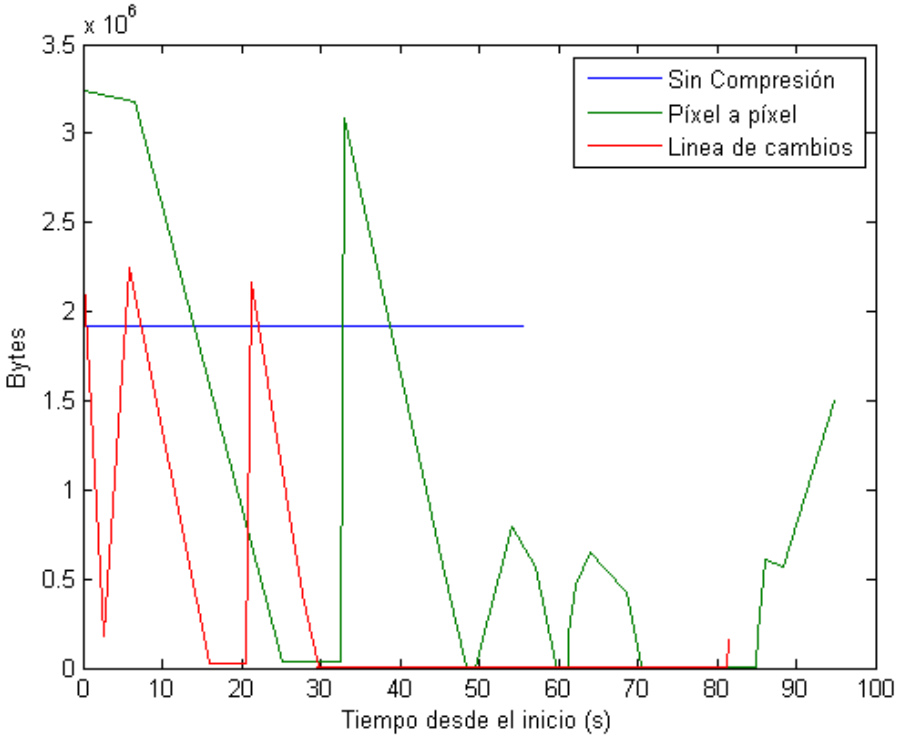

Figura 5.7: Tamaño de las imágenes para los tres algoritmos.

en comparación con los 22 ms conseguidos en Playout Delay en OnLive, nosotros nos quedamos en aproximadamente 60 segundos.

Aunque estos resultados son esperanzadores, no podemos olvidar que las métricas dependen mucho de los juegos. Para una comparativa más correcta habría que hacer las pruebas con los mismos juegos que utilizan en su estudio.

En las siguientes gráficas hacemos una comparativa entre 3 de los algoritmos de compresión utilizados. Los tres algoritmos son:

- Sin compresión
- Cambios píxel a píxel
- Cambios en línea

En la gráfica 5.7 se puede observar la evolución respecto al tiempo del tamaño del mensaje. En la gráfica 5.8 se puede ver la evolución del tiempo de codificación respecto al tiempo.

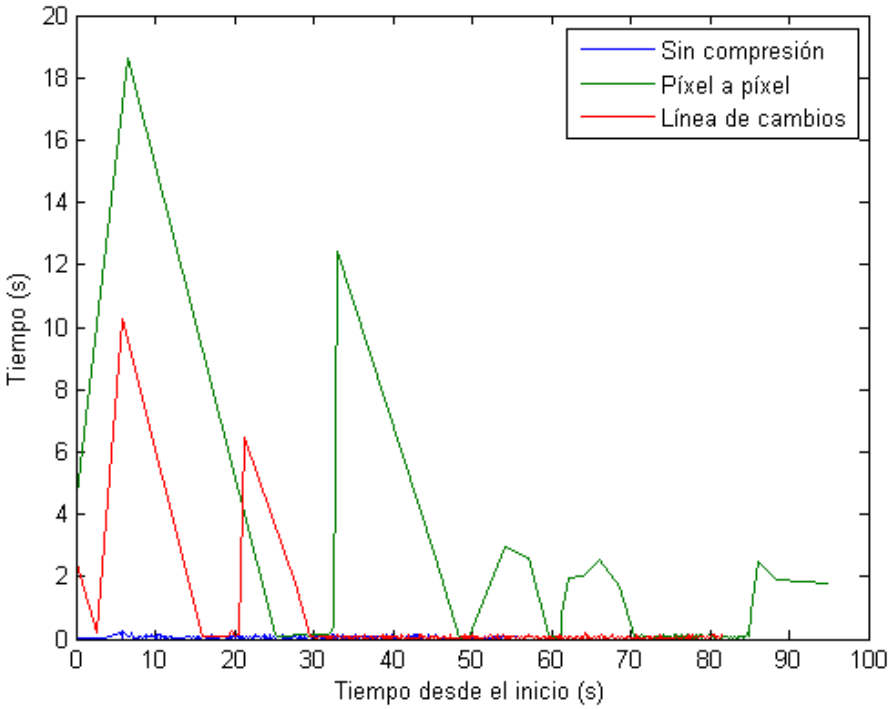

Figura 5.8: Tiempo de proceso de las imágenes para los tres algoritmos.

Al enviar la imagen completa, no comprimir mantiene un tamaño constante bastante alto. Aun así, tiene una tasa de codificación muy baja. Las variaciones que se ven se deben a la necesidad de cómputo del programa principal. Por otro lado, los cambios por punto generan mensajes muy grandes en casos en que haya muchos cambios en la superficie, y un gasto en memoria y cómputo que dispara los tiempos de codificación. En menor medida ocurre lo mismo con los cambios en línea, pero estos se adaptan mejor.

De estos resultados podemos deducir que la mejor opción es el uso de diversos algoritmos, sacando el mayor partido a todos ellos. Una opción con los algoritmos actuales es procesar tan solo hasta cierto número de cambios. Si existen más cambios, enviar la imagen completa. Si no, usar la lista de cambios. De esta manera no tendríamos tanto gasto en memoria y los procesos serían mucho más rápidos, a cambio de mandar unos pocos mensajes grandes.

Capítulo 6

Conclusión

El ámbito de este proyecto está en auge. Los grandes desarrolladores de software y hardware del sector están haciendo grandes inversiones en esta tecnología. Un estudio profundo y el desarrollo de una herramienta potente es de gran interés desde el punto de vista académico. REMIRTA permitirá a profesores y alumnos ahondar en los procesos relativos a estas tecnologías, y su uso en la industria.

Pero REMIRTA es todavía una prueba de concepto y tiene un gran futuro por delante. Con muchos aspectos a perfeccionar, y otros tantos abiertos a la investigación, REMIRTA es el primer paso al desarrollo de una plataforma muy potente.

En lo relativo a REPRO, las simulaciones arrojan resultados bastante prometedores. Uno de los aspectos importantes es que para conseguir los mejores resultados hay que hacer uso de un conjunto de los diferentes algoritmos de compresión. De todas formas, hay que profundizar en la manera de optimizar estos algoritmos y de crear algoritmos nuevos que permitan mejorar los tiempos de respuesta y la calidad del servicio.

Capítulo 7

Trabajo futuro

Este proyecto tiene grandes posibilidades de continuidad. Debido al poco tiempo disponible no se ha podido profundizar en otros temas de gran importancia. A continuación se muestran diversas líneas por las que continuar el proyecto.

Uno de los grandes atractivos de las aplicaciones multimedia es el sonido. Desgraciadamente, el proyecto, en esta versión, no tiene posibilidad de emisión de audio. Un posible trabajo futuro es adaptar el protocolo para la inclusión de audio.

El sistema está basado ahora mismo en Python, usando las librerías de PyGame. Tal y como está gestionado, es necesario modificar el código de la aplicación (en muy poca extensión) para que use nuestro sistema, lo cual limita su uso. Un posible trabajo futuro es portar el código para convertir nuestra aplicación en un driver de SDL, y así poder conectarlo transparentemente a la aplicación. Como ya hemos comentado previamente, las similitudes entre la estructura de PyGame y la de SDL ayudarán a esta labor.

Se puede trabajar en nuevos sistemas de compresión de flujo más eficientes tanto en tiempo como en tamaño. Además, se puede hacer uso de algoritmos y hardware más paralelos para mejorar el rendimiento de los sistemas ya propuestos.

Otro aspecto de mejora es la creación de una interfaz gráfica más completa para el cliente. Deberíamos permitir al cliente conectarse a distintos servidores, elegir un juego de ese servidor, y modificar parámetros como la profundad de color, la resolución, o la tasa de envío.

Bibliografía

[1] Yeng-Ting Lee, Kuan-Ta Chen, Han-I Su and Chin-Laung Lei,"Are All Games Equally Cloud-Gaming-Friendly? An Electromyographic Approach", Proceedings of IEEE/ACM NetGames 2012, Nov, 2012.

[2] Kuan-Ta Chen, Yu-Chun Chang, Po-Han Tseng, Chun-Ying Huang and Chin-Laung Lei, "Measuring The Latency of Cloud Gaming Systems", Proceedings of ACM Multimedia 2011, Nov, 2011.

[3] Documentación de PyGame, http://www.pygame.org

[4] Documentación de RFB, http://www.realvnc.com

[5] Documentación MPEG, http://mpeg.chiariglione.com

[6] Sitio web de OnLive, http://www.onlive.com

[7] Sitio web de Gaikai, http://www.gaikai.com

[8] Sitio web de Ubitus, http://www.ubitus.com

[9] Sitio web de GamingAnywhere, http://www.gaminganywhere.com

[10] Noticia: http://www.develop-online.net/news/44479/E3-2013-Gaikai-to-start-streaming-PS3-games-on-PS4-in-2014

Apéndice A

Métodos de compresión de flujo

A continuación se pasa a explicar los métodos usados.

A.1. Sin compresión

El método más simple para crear el flujo es enviar las imágenes completas. Aun siendo este método el más eficiente en cómputo, requeriría un ancho de banda demasiado alto. Por esto, se necesita comprimir de alguna manera el flujo.

A.2. Detección de cambios a nivel de píxel

Se compara píxel a píxel el último marco enviado con el próximo a enviar.

A.3. Detección de cambios en línea

Se compara el último marco enviado con el próximo a enviar. Al encontrar un cambio, se comprueba si el siguiente píxel también presenta un cambio. De ser cierto, se sigue comprobando hasta que se llega al máximo número de píxeles por línea, al final de la imagen, o se encuentra un píxel sin cambio.

Este sistema permite que un algoritmo lineal pueda recorrer la imagen aprovechando la localidad espacial.

A.4. Detección de cambios en línea discontinua

Este sistema es muy similar a la detección de cambios a nivel de línea. La variación es que se permite incluir píxeles sin cambio, siempre y cuando se mantenga un porcentaje de cambio en el total de la línea.

Con este sistema podemos aunar más cambios en una línea, teniendo menos sobrecarga en espacio.

A.5. Detección de movimientos de tipo desplazamiento

En muchos juegos tenemos un movimiento de desplazamiento de la pantalla. De esta manera, la pantalla tiene muchos cambios, pero simplemente desplazando el marco, reducimos en gran medida esos cambios. Para resolver esto, proponemos este método.

Comparamos en paralelo una serie de desplazamientos del último marco enviado con el próximo marco a enviar, usando cualquiera de los métodos anteriores. A estos, le sumamos la inclusión de los bordes que no comprueba el desplazamiento. Una vez tenemos el tamaño de los distintos mensajes potenciales, elegimos el de menor tamaño, y enviamos éste.

A.6. Cambio significativo

Este añadido se puede utilizar con cualquiera de los otros métodos. Se considera que un píxel no presenta cambio si no supera un umbral de cambio. Si lo supera, se ha producido un cambio significativo en el color, y se considera que el pixel ha cambiado.

A.7. Cambio significativo robusto

En el sistema de cambio significativo, una gradación de color pasaría desapercibida. Al entrar varios marcos sin cambios significativos entre ellos no se considera

cambio, pero sí existe un cambio a nivel global. Para esto se propone un sistema más robusto, que en vez de copiar la imagen enviada sobre la superficie de la última enviada, aplica los cambios que ha enviado al cliente. De esta manera es capaz de reconocer cambios globales.

Apéndice B

Mensajes en REPRO

Como ya se ha dicho en el cuerpo del proyecto, REPRO intenta reducir al mínimo el tamaño de los mensajes. Para conseguir esto, trabajamos a nivel de bit. De esta manera, reducimos aún más el tamaño del mensaje. Los distintos mensajes se explican a continuación.

B.1. Mensaje de envío de eventos

Este mensaje se usa para retransmitir las acciones del cliente hacia el servidor. Estas acciones se basan en los eventos que gestiona PyGame. PyGame gestiona 15 eventos predefinidos, aparte de otros eventos que pueda generar el programador. En total admite hasta 32 tipos distintos de eventos, por lo que se ha decido codificar cada evento usando 5 bits para su tipo.

Además de esto, los distintos tipos tienen diversos atributos. Por ejemplo, el evento de KeyDown, tiene los atributos que definen la tecla pulsada, los modificadores activo (R_Shift, L_Ctrl, etc...) y el carácter Unicode generado. Es por esto que la codificación que se ha seguido es compleja.

El mensaje está compuesto por:

- 2 bits que designan el tipo de mensaje (Mensaje de envío de eventos)
- 11 bits para designar el tamaño del mensaje en bytes
- 3 bits para mostrar cuántos bits vacíos hay al final del mensaje
- El conjunto de eventos codificado según el tipo

- Tantos bits de relleno como hagan falta para completar el último byte

A continuación explicamos la codificación de cada uno de los tipos de eventos.

B.1.1. QUIT

El evento QUIT surge cuando se cierra la ventana. No tiene atributos así que sólo se envía su código de evento.

B.1.2. VIDEOEXPOSE

El evento VIDEOEXPOSE se activa cuando el "display" es modificado por otro programa (habitualmente por el gestor de ventanas) y tiene que ser redibujado. Al igual que QUIT, no tiene atributos, así que sólo se envía su código de evento.

B.1.3. ACTIVEEVENT

El evento ACTIVEEVENT se usa para definir si la interfaz está activada. Tiene un atributo "gain" usado para definir si el evento es de ganancia o de perdida. Se usa un bit. Además tiene una máscara de tres bits para definir qué ha ganado o perdido. El primer píxel define la ganancia o pérdida del ratón. Cuando el ratón sale de la pantalla, se considera que se ha perdido el ratón, y cuando vuelve a la pantalla, se considera que se ha recuperado. El segundo bit define si se ha ganado o perdido el foco del teclado. Cuando seleccionamos otra ventana, perdemos el foco del teclado, y cuando volvemos a seleccionarla, lo recuperamos. Por último, el tercer bit define si la pantalla está activa. Si la pantalla se minimiza, se considera que se ha perdido la activación. Cuando se restaura la ventana, se vuelve a ganar la activación.

B.1.4. KEYDOWN

Como explicamos previamente, el evento de pulsación de teclas tiene tres atributos. Primero usamos 16 bits para establecer el carácter Unicode. Luego usamos 10 bits para definir la tecla que estamos pulsando. Por último, usamos una máscara de 11 bits para cada una de las teclas modificadoras.

B.1.5. KEYUP

Se genera cuando se levanta una tecla del teclado. Se codifica igual que KEYDOWN, salvo en el código de evento, y en que no se incluye el carácter Unicode.

B.1.6. MOUSEBUTTONDOWN

Cuando se pulsa una tecla del ratón se genera un evento MOUSEBUTTONDOWN. Los atributos son las posición en el eje X, la posición en el eje Y, y los botones que se han pulsado. Para codificarlo hemos usado 10 bits para cada una de las posiciones, y una máscara de tres bits para los tres botones del ratón: izquierdo, central y derecho.

B.1.7. MOUSEBUTTONUP

Se genera cuando se suelta una tecla del ratón. Salvo el código de evento, se codifica igual que el evento MOUSEBUTTONDOWN.

B.1.8. MOUSEMOTION

Cuando el ratón se mueve por la pantalla, genera eventos que constan de 5 atributos: La posición final (en las coordenadas X e Y), el movimiento relativo que ha efectuado el ratón (en las coordenadas X e Y), y los botones pulsados.

Para su codificación se han usado:

- 10 bits para la posición en X
- 10 bits para la posición en Y
- 1 bit para el signo del movimiento relativo en X
- 10 bits para el movimiento relativo en X
- 1 bit para el signo del movimiento relativo en Y
- 10 bits para el movimiento relativo en Y
- Una máscara de 3 bits para los botones del ratón

B.1.9. VIDEORESIZE

Al modificar el tamaño de la pantalla se genera un evento VIDEORESIZE. Este evento tiene dos atributos, el nuevo ancho y el nuevo alto de la pantalla. Se ha codificado con 10 bits para cada atributo.

B.2. Mensaje de envío de imágenes

Todos los mensajes de imágenes comparten la siguiente estructura básica:

- 2 bits de identificación del mensaje
- 11 bits para denominar el marco
- 12 bits para el tamaño del mensaje
- La imagen

Los bits usados para denominar el marco podrán ser utilizados en versiones futuras para establecer un control sobre el orden de los marcos y posibles acciones ante pérdidas de marcos.

Según el tipo de compresión de flujo generaremos distintos mensajes. A continuación se detalla.

B.2.1. Sin compresión

Se mandan los píxeles sin ninguna clase de compresión. Cada píxel está definido por 4 bytes que corresponden a su color.

B.2.2. Sólo cambios

Se establecen 3 bits para contar los bits vacíos que hay al final del mensaje. Luego viene una lista de cambios, que están formados por 22 píxeles que codifican el píxel que ha cambiado y 4 bytes con el nuevo color. Tras la lista, se rellenan los bits vacíos hasta formar el último byte.

B.2.3. Cambios en línea

En los cambios por línea también se utilizan 3 bits para contar los bits vacíos al final del mensaje. Los cambios, sin embargo, están definidos por 22 bits que denominan la posición, 2 bits que muestran cuantos píxeles han cambiado en línea, y tantos grupos de 4 bytes como píxeles hayan cambiado.

B.2.4. Cambio por desplazamiento

Volvemos a utilizar 3 bits para contar los bits vacíos al final del mensaje. Usamos 2 bits más para definir hacia dónde ha sido el desplazamiento. Luego usamos tantos grupos de 4 bytes para transmitir el borde horizontal que hemos dejado. Después usamos tantos grupos de 4 bytes como sean necesarios para transmitir el borde vertical que hemos dejado. Por último añadimos los cambios con el mismo formato de cambios en línea.